https://doi.org/10.3311/PPee.17853

Periodica Polytechnica Electrical Engineering and Computer Science

Improved Runtime for the Synchronous Multi-door Filling

Attila Hideg[1*], Tamás Lukovszki[2], Bertalan Forstner[1]

[1] Department of Automation and Applied Informatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, H-1117 Budapest, Magyar tudósok krt. 2., Hungary
[2] Faculty of Informatics, H-1117 Budapest, Eötvös Loránd University, Pázmány Péter sétány 1/C., Hungary
[*] Corresponding author, e-mail: Attila.Hideg@aut.bme.hu

Received: 14 January 2021, Accepted: 14 June 2021, Published online: 30 November 2021

Abstract

In this paper, a particular type of dispersion is further investigated, which is called Filling. In this problem, robots are injected one by one into an a priori not known area and have to travel across until the whole area is covered. The coverage is achieved by a robotic team whose hardware capabilities are restricted in order to maintain low production costs. This includes limited viewing and communication ranges. In this work, we present an algorithm solving the synchronous Filling problem in $O((k + \Delta)\cdot n)$ time steps by n robots with a viewing range of 1 hop, where k is the number of doors, n is the number of vertices of the graph, and Δ is the maximum degree of the graph. This improves the best previously known running time bound of $O(k \cdot \Delta \cdot n)$. Furthermore, we remove the constraint from the previous algorithm that the door vertices need to have a degree of 1.

Keywords

filling, uniform dispersal, multi-robot system

1 Introduction

We consider a multi-robot system consisting of a team of homogenous robots. The robots are simple, cheap, and the team members collaboratively solve a common problem. The design and control of multi-robot systems are more complicated than a single robot system. However, multi-robot systems have several advantages over single robot systems. These advantages become apparent when they have to solve problems that can be fulfilled by multiple robots acting at the same time in the same area. Typical problems that these systems have to solve are exploration and coverage of an area, foraging, search and rescue, collaborative task planning and execution, etc…. They could also be used in dangerous environments for human health, such as environments contaminated by nuclear waste or toxic gases.

The concept for artificial swarms, coming from the wildlife, means several entities belonging to the same group. Such as bees or birds, members of a swarm belong together and do the same thing; for example, flocking, the collective movement of the entities. Reynolds [1] investigated the possibilities of nature-inspired algorithms. The paper described a simple algorithm for the flocking, and since its publication, artificial swarms have gained more and more attention.

Our previous work examined a particular dispersion problem called the Filling, which required robots to cover a previously unknown area represented by a graph. In [2], an algorithm called the Virtual Chain Method (VCM) was presented. In this paper, we further improve this algorithm, namely, we have made improvements in two specific ways. First, we modify the VCM algorithm, significantly improving the running time.

Furthermore, we remove a constraint of the graph (namely, the door degree having to be 1 is not required anymore; further explained in Section 2. As a result, the VCM algorithm could work in more areas than before.

2 Background and related work

The area where the robots operate is decomposed into small regions. These regions are connected to each other and can contain at most one robot. The area will be represented by an arbitrary graph whose vertices correspond to these regions, and the edges will be between neighboring regions (regions sharing a side). There are special vertices, called Doors, which will be the entry points for the robots, where they start exploring the area. We assume that the adjacent vertices are arranged in a fixed cyclic order for each vertex, which does not change during the dispersion (an example is shown in Fig. 1).

Cite this article as: Hideg, A., Lukovszki, T., Forstner, B. "Improved Runtime for the Synchronous Multi-Door Filling", Periodica Polytechnica Electrical Engineering and Computer Science, 2021. https://doi.org/10.3311/PPee.17853

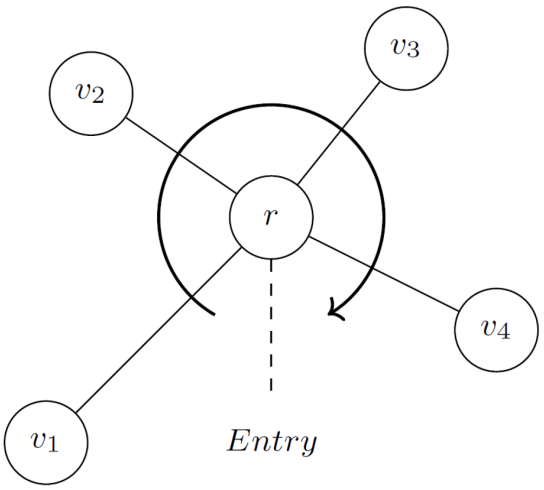

Fig. 1 Cyclic order of the nodes. Robot *r* (in the center) has several neighbors (v_1, v_2, v_3, v_4). The order depends on the Entry, which will be used as a pivot. Then the order will be in clockwise order

The robots are small agents capable of moving and sensing their surroundings. Their hardware is kept simple to maintain their cost-efficiency. The reason for this is that for the robots to be able to sense and communicate, additional resources have to be built into them. However, the lack of expensive hardware restricts their perception and communication. The communication and sensing range will be measured in hops (in the graph). Another typical restricting aspect is the size of their persistent memory, measured in bits, which is highly limited.

The robots act according to the Look-Compute-Move (LCM) model. This means the actions of the robots are decomposed into the following three phases: in the *Look* phase, they take a snapshot with their sensors, in the *Compute* phase, they perform their computations, and in the *Move* phase, they perform their actions (in this case, their movements). After the movement, the previous position of the robot will be referred to as its *Entry*.

The Filling problem, which is investigated in this paper, was introduced by Hsiang et al. [3]. This is a special type of dispersion where robots are placed at the door vertex one by one, and by the time the algorithm terminates, each of the vertices has to be occupied by a robot. When there is more than one door, the problem is called multi-door Filling or *k*-door Filling (where *k* is the number of doors). The algorithm terminates when each vertex of the graph is occupied by exactly one robot. I.e., *n* robots are required to fill a graph consisting of *n* vertices. In the Multiple Door Filling, it is assumed that there are enough robots to allow the placement of a new one whenever a Door vertex becomes empty.

Barrameda et al. [4] investigated the hardware requirements of the robots, searching for the lowest possible hardware requirement while still being able to solve the Filling problem. They presented a method for the standard and multi-door Filling with the goal of solving it by robots with constant visibility and communication range. In [5], they presented a method TALK, which required 1 hop visibility and 1 hop communication range, while MUTE required 6 hops visibility range and no communication. Both algorithms worked in orthogonal areas (i.e., grid-like areas where each cell had at most 4 neighbors).

The authors presented an algorithm further improved in [6] where they required 1 hop visibility and no communication. This method worked in orthogonal areas and required a shared knowledge of directions (North, East, South, West). In [2], it was further improved to remove the requirement of orthogonal areas: it could solve the Filling in arbitrary areas at the cost of runtime and memory.

Recently, several versions of the Filling problem have been studied regarding the capabilities of the robots, e.g., allowing or forbidding collision, communication, having various visibility ranges, using whiteboards at the vertices of the graph.

Augustine and Moses [7] introduced a version of the dispersion problem, where more than one robot can be at the same vertex (collision is allowed). The robots can only see the robots at the same vertex and are able to communicate with them. In [7], they presented two algorithms, one with a running time of $O(m \cdot n)$ rounds and $O(\log n)$ memory size of the robots, where m is the number of edges in the graph; and another algorithm with $O(m)$ running time and $O(n \cdot \log n)$ memory. Kshemkalyani et al. [8] allowed global communication between the robots and they presented an algorithm with $O(\log(\max(k, \Delta)))$ bits of memory and $O(\min(m, k\Delta)))$ running time, where Δ is the maximum degree of the graph. They also presented another algorithm with $O(\max(D, \Delta \log k))$ bits memory requirement and $O(\max(\Delta, k)\Delta(D + \Delta))$ runtime, where D is the diameter of the graph. In [9], the authors solve this problem without global communication with $O(k, \log \Delta)$ bits memory and $O(m)$ running time. In [10], these bounds are improved to $O(\log n)$ bits memory and $O(\min(m, k\Delta) \cdot \log k)$ running time. They also provided a lower bound of $O(\log n)$ for the memory requirement in this model.

Dereniowski et al. [11] investigated the memory and time requirements for the exploration problem, where the robots need to visit each vertex but do not need to settle there. Their synchronous algorithm required $O(k \cdot \log \Delta)$ bits of memory

and $O(D)$ runtime, however, they required n^D robots. They also investigated the trade-off between exploration time and team size. Czyzowicz et al. [12] studied the complexity of exploration algorithms in the continuous Euclidean space in both the bound and the unbound visibility model.

Amir and Bruckstein [13] considered the single Door case of the Filling problem in orthogonal areas using robots with a visibility range of 2 hops. The aim was to minimize the travel of the individual robots. They presented an algorithm of $O(n)$ running time. In scenarios where the movement uses much more energy than staying in place, this minimizes the energy requirements. However, there are certain scenarios where this is not the case, e.g., flying drones use nearly the same energy for hovering and moving as in [14]. In [15], scenarios with crash-prone robots were also considered.

2.1 Virtual Chain Method

The Virtual Chain Method presented in [2] was designed to minimize the hardware requirements to 1 hop visibility, $O(\Delta \cdot \log k)$ bits of memory, and no communication or other hardware (e.g., global or local positioning hardware). A short description of the algorithm:

Among the robots, there was a distinct one, which was the leader robot. Its main task was to go to vertices that were never occupied by a robot (named *unvisited* vertices). The rest of the robots had follower roles and followed that leader robot. This way, a non-visible chain was formed, with the leader in the front, hence the name virtual chain method. However, several tasks had to be solved. First, the robots had to cover the whole area, i.e., when the leader stuck, the next robot had to become the new leader and move in a different direction. Second, the robots had to follow the robot which has been previously placed (called their *predecessor*), even when those robots were outside of their visibility range (as they had a 1 hop visibility). Third, the robots had to avoid collisions.

These were addressed by introducing a unique round structure: an LCM-cycle was called a *step*, and a round consisted of Δ steps (see Fig. 2.). There were two different rounds: odd and even. The robots started with an odd round, and then the even and odd rounds were alternating during the algorithm. In its odd round, a robot is only allowed to observe its surroundings, while in the even round, it was allowed to move. The robots had to time their movement in this round structure, i.e., the step they utilized to move had a meaning. In their even round, they moved to the i^{th} neighbor (in the cyclic order) in the i^{th} step.

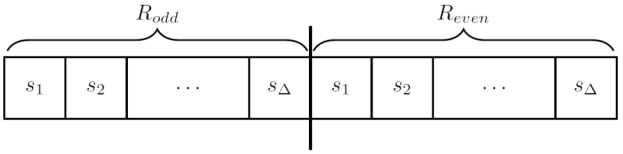

Fig. 2 The structure of the rounds. The rounds (denoted by R_{odd}, R_{even}) consist of Δ consecutive steps in a fixed order

Therefore, the robot observing it knows which direction it went. When a robot is in its odd round, it counts the number of steps to know which direction the predecessor goes.

The round structure solves all problems in the following way:

- The first issue is solved by using the timing of the movement. If the predecessor robot did not move, it implied that it could not move. Therefore, the robot whose predecessor did not move became the new leader and moved to a different direction.
- The second issue (namely, to follow the predecessor even when it moves out of visibility range) is possible as the predecessor robot implicitly signals its movement direction by timing its movement.
- The third issue (collision avoidance) was avoided by the alternating rounds. If a vertex is not unvisited, then there was a robot who occupied it. When that robot left it in its even round, the next robot has its odd round. In the next round, that robot will have its even round and move there. Therefore, the robots knew which cells are not unvisited by observing them for two rounds.

However, there were two constraints:

- First, the robots have to know the cyclic order. This was implicitly the same as they entered from the same direction (same vertex) in each movement. However, at the first vertex (the door vertex), depending on the direction they were facing during the placement, it might be a different order (i.e., the robots had no means to know which neighbor is the first in the cyclic order). This was addressed by having the door vertex only 1 degree (as a door-step). Then from that point, they knew which was their entry vertex, which was the basis of the cyclic order.
- Secondly, when there were multiple chains (in the multiple door case), the two chains could have collided. This was addressed by having distinct steps for each chain (see Fig. 3). However, having k doors meant that the runtime was increased by a factor of k. This can be 'perceived' as the overhead coming from the coordination of the chains.

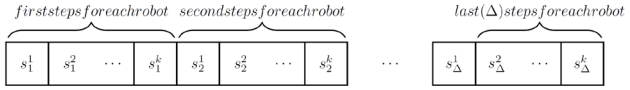

Fig. 3 Round structure for the multiple door filling. In s_j^i the robots from the i^{th} Door perform their j^{th} step

More details and proof of correctness can be found in [2].

In this paper, these constraints are removed by introducing a novel method.

3 Method

There are two modifications in the VCM method, which is based on the same principle to improve the algorithm in two ways: first, the runtime factor of $O(k \cdot \Delta \cdot n)$ is reduced to $O((k + \Delta) \cdot n)$, where n is the size of the area; then, the constraint of having 1-degree doors is removed.

A new rule is introduced, which is called *back and forth movement*. This is a complex movement with two phases:

- the robot moves from its vertex v to a neighbor v' in step s_i.
- the robot moves from its new position v' to v (its original one) in step s_{i+1}.

During this movement, it is important to 'save' their state before the movement and restore it afterward. The back and forth movement is still simple enough to be performed by the cheap robots with low computational capabilities; moreover, it does not increase the hardware requirements.

3.1 Fast Virtual Chain Method

The factor of k in the $O(k \cdot \Delta \cdot n)$ runtime comes from the added steps for collision avoidance. The followers would not collide with other robots as each follower only follows its predecessor, and the predecessor is unique among followers; followers cannot collide with each other. The only collision can happen if a Leader robot would move to a vertex simultaneously with another robot. The leader only moves to unvisited vertices, therefore, the other robot cannot be a follower, i.e., only two leaders can collide. This could happen if two leaders would move to the same vertex, and to prevent their collision, the chains had different timeslots (see Fig. 3).

The new method, called Fast Virtual Chain Method (FVCM), will utilize the back and forth movement in the following way: each leader will perform a back and forth movement first to 'reserve' its target vertex (before showing which way to go), then each chain simultaneously perform their movements. This significantly reduces the runtime, as instead of $k \cdot \Delta$ steps, only $k + 1 + \Delta$ will be required for a round. The following paragraph contains a detailed description.

Label the doors D_i and let L_i be the current leader from D_i. The new rule for a leader L_i will be to perform a back and forth movement in step s_i towards its first unvisited neighbor. E.g., in s_1, L_1 will move to v and moves back in s_2 to its original position. During the Look phase of s_2, each robot neighboring to v will remove v from their list of unvisited neighbors. This will prevent their collision with L_1. In s_2, L_2 will move to its current first unvisited neighbor (which must be other than v). This is repeated until L_k performs its back and forth movement in s_k (and moves back in s_{k+1}). Afterward, in the next Δ steps, the robots perform their original VCM algorithms as if there would be only a single door.

The labeling of the steps will be: $S_{L_i}(i \leq k + 1)$ for the back and forth steps, and s_j $(j \leq \Delta)$ for the movement steps (see Fig. 4).

3.2 Analysis

Lemma 1. *In the Fast Virtual Chain Method, there are no collisions.*

Proof. The robots would collide if they moved to the same vertex v at the same time. The possible states of those two robots can be one of the following: both are followers, both are leaders, or one of them is a follower, and the other one is a leader.

In the first case, when both are followers, the collision is not possible as the followers follow a different robot. Therefore, when they move, they choose a distinct vertex as their target.

In the second case, as the leader can only move to an unvisited vertex, it cannot move to one which had been occupied before. However, the vertices where the followers move were occupied previously (by their predecessor) and are not unvisited vertices. Therefore, the leader cannot move there and cannot collide with a follower.

The third case, when two leaders would move to the same target vertex, had the potential to cause a collision. This is solved by forcing the leaders to 'reserve' their target before the actual movement (which includes the direction signaling). Let L_i and $L_j(i < j)$ be two leaders who would

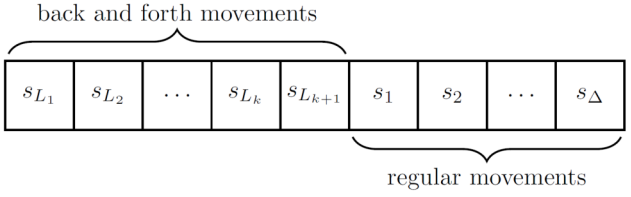

Fig. 4 New rounds structure for the FVCM. In the first $k + 1$ steps (S_{L_i} the leaders perform a back and forth movement. In the next Δ steps (S_j) they perform their original movements

collide by both choosing v as their target vertex. However, the two leaders L_i and L_j would move to v in different steps (S_{L_i} and S_{L_j}). After the former step S_{L_i} the vertex v has already been removed from the list of unvisited vertices.

An example can be seen in Fig. 5 (a), where two robots, L_1 and L_2 could potentially collide if both would go to v at the same time. However, L_1 moves to v and 'reserves' it in S_{L_1} (see Fig. 5 (b)). When L_2 detects L_1, it will remove v from its list of unvisited vertices. Therefore, collisions are not possible. In the next step S_{L_2}, L_1 performs its backward movement while L_2 chooses another vertex as its target (Fig. 5 (c)).

Theorem 1. *The FVCM algorithm fills an arbitrary connected graph in rounds by silent robots with a visibility range of 1 hop and bits of memory.*

Proof. The visibility and communication requirements are the same as in [2]. However, the back and forth movement modifies the memory requirements. The robots must store the round length, which is reduced from $\Delta \cdot k$ to $\Delta + k + 1$. Therefore, the memory requirement of the robots is decreased from $O(\Delta \cdot \log k)$ to $O(\Delta + \log k)$ bits. There is a requirement of 1 additional bit in order to store when it has to perform a back movement (during its back and forth movement). This does not change the $O(\Delta + \log k)$ memory requirement.

The runtime improvement follows from the new length of the rounds. Each round consists of $k + 1 + \Delta$ steps, and the robots move in every second round (their even round), and a new robot is placed at the door in every third round. As each round consists of $k + 1 + \Delta$ steps, it takes $O((k++\Delta) \cdot \mathrm{n})$ steps to place n robots.

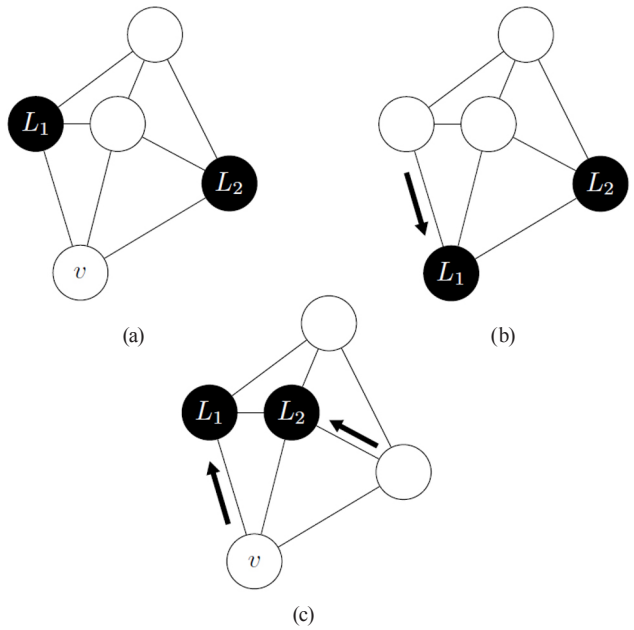

Fig. 5 An example for the back and forth movements of two leaders, L_1 and L_2. In (a) they do not see each other but select v as their target. In (b) L_1 moves to v, 'reserving', then in (c) L_2 must go to a different target.

3.3 Higher degree doors

The second improvement of the VCM eliminates the constraint of having 1-degree door vertices in [2]. This was to ensure that the robots know the first direction they are going, and then the cyclic order is known. Thus, to have higher degree door vertices, the only task to be solved is to make sure newly placed robots know where their predecessor is.

For the newly placed robot, the back and forth movement will be utilized to find its predecessor. Similarly to the improved runtime, $k+1$ additional steps will be added. In these steps, for each door D_i, $i = 1 \ldots k$, and step S_{L_i} the predecessor of the robot in D_i will signal its position by performing a back and forth movement. The robot of the i^{th} chain (chain originating form D_i) uses S_{L_i} for forward and $S_{L_{i+1}}$ for backward movement. No other robots can move in these $k+1$ steps. This can happen in the round the predecessor would move to its next vertex (when it is unoccupied).

With this extension, the robot on D_i will learn where its predecessor is, as in S_{L_i} the only robot allowed to move is that predecessor.

3.3.1 Combination with the faster runtime

The two improvements (faster runtime and higher degree doors) can be combined by simply allocating 2 steps to each chain (S_{L_i} and $S_{L_{i'}}$).

- In the first step (S_{L_i}), the robot, which is showing the direction to the robot at the door vertex, moves (starts a back and forth movement).
- In the second step ($S_{L_{i'}}$) the robot moves back, and in the same step, the leader of that chain starts to perform its back and forth movement (which is finished in the third step but that does not interfere with the other chains).

Note that after the first $2k$ steps, an additional step is required for the k^{th} leader to perform the return step of its back and forth movement. This will yield a round length of $2k + 1 + \Delta$ steps. The runtime is still $O((k+\Delta) \cdot n)$.

4 Simulation

To validate the presented algorithms, simulations were carried out to verify both the correctness of the algorithms and the runtime improvement. The graphs were created using the following method:

- *Graph with 1-degree doors:* for a graph with n vertices and k doors, n-k points were randomly placed, and using these points, a Delaunay triangulation is created. Then, k vertices were randomly selected from them, then the door vertices were added to them.
- *Graph with multiple degree doors:* for a graph with n vertices and k doors, n points were randomly placed, and using these points, a Delaunay triangulation is created. Then, k vertices were randomly selected to become doors.

4.1 Validation of runtime

The first set of simulations validated the theoretical results, namely that the presented methods cover the whole area, and there are no collisions. The simulations measured the runtime of these algorithms in the 1, 2, and 3-door cases. The graphs were created using the previously described methods with sizes from 1 to 200.

The simulation results can be seen in Fig. 6. The vertical axis shows the number of LCM-cycles (or steps) to fill the area completely, and the horizontal axis shows the number of doors (k).

The runtime was within the $O((k+\Delta)\cdot n)$ bound. The lines are not straight, which is caused by the slight variation in Δ (the maximum degree Δ was between 8 and 12).

4.2 Runtime improvement

The second sets of simulations tested the runtime improvement of the FVCM (compared to the original VCM) for the multiple Door case. The same graphs were created with 200 vertices and k doors, from $k=1$ to 99, and both the VCM and the FVMC were executed on them. (for $k>100$, the runtime will become constant, since when half of the vertices are door vertices, after moving from those in the first round, the other vertices will become occupied, and as soon as a new robot is placed at the doors, the graph will be filled)

Fig. 7 shows the results (again, the vertical axis shows the number of steps; the horizontal axis shows the number of doors).

An interesting result is that in the original VCM, adding doors increases the runtime. This might be the result of the increased round length. To further investigate this, the number of required rounds as we assume that the total rounds are decreasing but their length increase causing the increased total runtime.

In Fig. 8, it is clearly visible that the rounds are decreasing as expected in both cases. (Note that the algorithms are based on the same principle, the robots move in every second round; therefore, the number of rounds are the same). However, the length of the rounds is decreased from $\Delta\cdot k$ to $\Delta+k+1$.

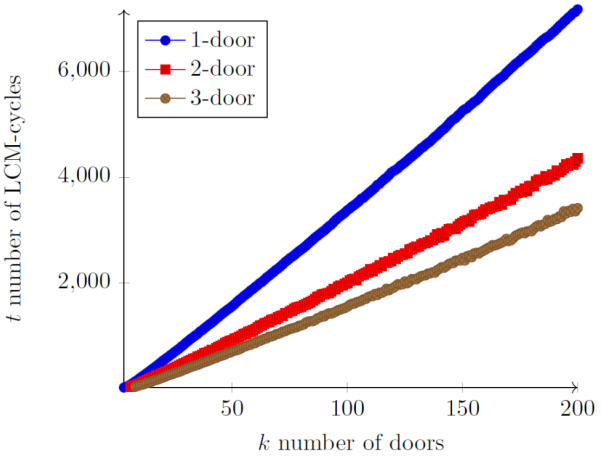

Fig. 6 FVCM performance for the 1, 2, and 3-Door cases

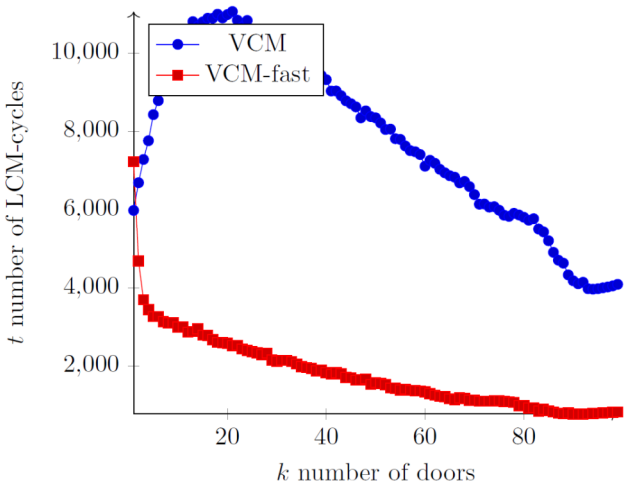

Fig. 7 Comparison of the runtime of the VCM and FVCM

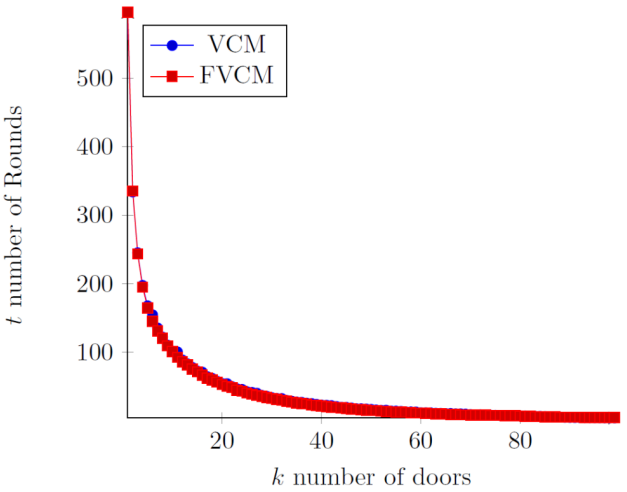

Fig. 8 Comparison of the number of Rounds for the VCM and FVCM

Table 1 The round lengths, number of rounds, and number of steps during the simulation for $k \in \{1,2,3,4,5,10,20\}$ and $\Delta = 10$.

k	Δ	VCM round length	VCM number of rounds	VCM number of steps	FVCM round length	FVCM number of rounds	FVCM number of steps
1	10	10	596	5960	12	596	7152
2	10	20	333	6660	13	333	4329
3	10	30	245	7350	14	245	3430
4	10	40	197	7880	15	197	2955
5	10	50	167	8350	16	167	2672
10	10	100	100	10000	21	100	2100
20	10	200	52	10400	31	52	1612

Table 1 shows the number of rounds and steps for the VCM and FVCM for several k and Δ parameters.

5 Summary

In this paper, we improved our previous solutions for the Filling problem. We proposed an extension of the algorithm, which was simple enough to be performed by cheap robots with low computational power. Utilizing this rule, we significantly lowered the running time of the algorithm without additional hardware requirements. Furthermore, we removed the constraint of our previous algorithm in [2], namely that the Door vertices needed to have a degree of 1, causing the algorithm to be able to operate in more general types of areas immediately. We backed these results with simulations. It remains an open question if the running time can be further improved.

Acknowledgement

The work presented in this paper has been carried out in the frame of project no. 2019-1.1.1-PIACI-KFI-2019-00263, which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.1. funding scheme.

References

[1] Reynolds, C. W. "Flocks, herds and schools: A distributed behavioral model", ACM SIGGRAPH Computer Graphics, 21(4), pp. 25–34, 1987. https://doi.org/10.1145/37402.37406

[2] Hideg A., Lukovszki T., Forstner B. "Filling Arbitrary Connected Areas by Silent Robots with Minimum Visibility Range", In: Gilbert, S., Hughes, D., Krishnamachari, B. (eds.) Algorithms for Sensor Systems, Springer, Cham, Switzerland, 2019, 193–205. https://doi.org/10.1007/978-3-030-14094-6_13

[3] Hsiang, T.-R., Arkin, E. M., Bender, M. A., Fekete, S. P., Mitchell, J. S. B. "Algorithms for Rapidly Dispersing Robot Swarms in Unknown Environments", In: Boissonnat, J. D., Burdick, J., Goldberg, K., Hutchinson, S. (eds.) Algorithmic Foundations of Robotics V, Springer, Berlin, Germany, 2004, pp. 77–93. https://doi.org/10.1007/978-3-540-45058-0_6

[4] Barrameda, E. M., Das, S., Santoro, N. "Deployment of Asynchronous Robotic Sensors in Unknown Orthogonal Environments", In: Fekete, S. P. (ed.) Algorithmic Aspects of Wireless Sensor Networks, Springer, Berlin, Germany, 2008, pp. 125–140. https://doi.org/10.1007/978-3-540-92862-1_11

[5] Barrameda, E. M., Das, S., Santoro, N. "Uniform Dispersal of Asynchronous Finite-State Mobile Robots in Presence of Holes", In: Flocchini, P., Gao, J., Kranakis, E., Meyer auf dr Heide, F. (eds.) Algorithms for Sensor Systems, Spriner, Berlin, Germany, 2014, pp. 228–243. http://doi.org/10.1007/978-3-642-45346-5_17

[6] Hideg, A., Lukovszki, T. "Uniform Dispersal of Robots with Minimum Visibility Range", In: Fernández Anta, A., Jurdzinski, T., Mosterio, M., Zhang, Y. (eds.) Algorithms for Sensor Systems, Spinger, Cham, Switzerland, 2017, pp. 155–167. https://doi.org/10.1007/978-3-319-72751-6_12

[7] Augustine, J., Moses, W. K. "Dispersion of Mobile Robots: A Study of Memory-Time Trade-offs", In: Proceedings of the 19th International Conference on Distributed Computing and Networking (ICDCN'18),Varanasi, India, 2018, Article No. 1. https://doi.org/10.1145/3154273.3154293

[8] Kshemkalyani, A. D., Molla, A. R., Sharma, G. "Dispersion of Mobile Robots in the Global Communication Model", In: Proceedings of the 21st International Conference on Distributed Computing and Networking (ICDCN'2020), Kolkata, India, 2020, Article number: 12. https://doi.org/10.1145/3369740.3369775

[9] Kshemkalyani, A. D., Ali, F. "Efficient dispersion of mobile robots on graphs", In: Proceedings of the 20th International Conference on Distributed Computing and Networking (ICDCN'19), Banglore, India, pp. 218–227. https://doi.org/10.1145/3288599.3288610

[10] Kshemkalyani A. D., Molla A. R., Sharma, G. "Fast Dispersion of Mobile Robots on Arbitrary Graphs", In: Dressler, F., Scheideler, C. (eds.) Algorithms for Sensor Systems, Springer, Cham, Switzerland, 2019, pp. 23–40. https://doi.org/10.1007/978-3-030-34405-4_2

[11] Dereniowski, D., Disser, Y., Kosowski, A., Pająk, D., Uznański, P. "Fast collaborative graph exploration", Information and Computation, 243, pp. 37–49, 2015. https://doi.org/10.1016/j.ic.2014.12.005

[12] Czyzowicz, J., Ilcinkas, D., Labourel, A., Pelc, A. "Worst-case optimal exploration of terrains with obstacles", Information and Computation, 225, pp. 16–28, 2013. https://doi.org/10.1016/j.ic.2013.02.001

[13] Amir, M., Bruckstein, A. M. "Minimizing Travel in the Uniform Dispersal Problem for Robotic Sensors", In: Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'19), Montreal, Canada, 2019, pp. 113–121. https://dl.acm.org/doi/10.5555/3306127.3331682

[14] Saska, M., Baca, T., Thomas, J., Chodoba, J., Preucil, L., Krajnik, T., Faigl, J., Loianno, G., Kumar, V. "System for deployment of groups of unmanned micro aerial vehicles in GPS-denied environments using onboard visual relative localization", Autonomous Robots, 41(4), pp. 919–944, 2017. https://doi.org/10.1007/s10514-016-9567-z

[15] Amir, M., Bruckstein, A. M. "Fast Uniform Dispersion of a Crash-prone Swarm", presented at Robotics: Science and Systems 2020, Corvalis, OR, USA, July, 12–16, 2020. https://doi.org/10.15607/rss.2020.xvi.017